VERİ SIKIŞTIRMA

İstatistiksel Yöntemler-5

Prof.Dr. Banu DİRİ

Aritmetik Kodlama -Arithmetic Coding

- Sembollerin olasılık değerleri ikinin negatif kuvvetlerine sahip ise Shannon Fano iyi sonuç veriyordu
- Huffman Yöntemi, Shannon Fano Yönteminden daha iyi sonuç veriyordu
- Bir sembolün olasılığı 0,4 ise bu sembole atanan bit $-\log_2 0{,}4 \approx 1{,}32$ (1 veya 2 bit)
- Aritmetik kodlamada, tek bir sembole değil de bütün input stream'e bir kod atanır
- Atanan bu kod değeri [0, 1) aralığında bir sayı olur
- Eğer kod, «9746509» ise bu «0, 9746509» olarak yorumlanmalıdır. Ancak, «0» output stream yazılmaz
- İki pass'de çalışan bir yöntemdir. İlk pass'de sembollerin kullanım sıklıkları çıkarılır, ikinci pass'de ise sıkıştırma işlemi gerçekleştirilir

Örnek

Olasılık değerleri sırası ile P1 = 0.4, P2 = 0.5 ve P3 = 0.1 olan üç sembol olsun. Sembollerimiz a1, a2 ve a3

[0, 1) aralığını, üç sembole bölmeye çalışalım.

a1 → [0, 0.4)
a2 → [0.4, 0.9)
a3 → [0.9, 1)

```
NewHigh:=OldLow+Range*HighRange(X);
NewLow:=OldLow+Range*LowRange(X);
```

«a2a2a2a3» stringini Aritmetik Kodlama ile nasıl kodlarız ?

«a2a2a2a3»

```
NewHigh:=OldLow+Range*HighRange(X);
NewLow:=OldLow+Range*LowRange(X);
```

a1 → [0, 0.4)
a2 → [0.4, 0.9)
a3 → [0.9, 1)

Okunan ilk **a2** ile [0, 1) aralığı daraltılmaya başlanır

Low = 0 + (1 - 0) * 0.4 = 0.4
High = 0 + (1 - 0) * 0.9 = 0.9

İkinci **a2** ile [0.4, 0.9) aralığı aynı yolla daraltılmaya çalışılır

Low = 0.4 + (0.9 - 0.4) * 0.4 = 0.6
High = 0.4 + (0.9 - 0.4) * 0.9 = 0.85

Üçüncü **a2** ile [0.6, 0.85) aralığı aynı yolla daraltılmaya çalışılır

Low = 0.6 + (0.85 - 0.6) * 0.4 = 0.7
High = 0.6 + (0.85 - 0.6) * 0.9 = 0.825

Dördüncü **a3** ile [0.7, 0.825) aralığı aynı yolla daraltılmaya çalışılır

Low = 0.7 + (0.825 - 0.7) * 0.9 = 0.8125
High = 0.7 + (0.825 - 0.7) * 1 = 0.825

Çıkış bu aralık içerisinde bir sayıdır

Kurallar

1. [0, 1) aralığı ile işleme başlanır
2. Input stream'de yer alan her «s» sembolü için aşağıdaki iki adım tekrarlanır
   - Geçerli aralık, alt aralıklara bölünür ki, bu alt aralıkların uzunluğu sembollerin olasılıkları ile orantılıdır
   - «s» için alt aralık seçilir ve yeni geçerli aralık olarak tanımlanır
3. Bütün input stream bu yolla işlendiği zaman, çıkış geçerli aralık arasında kalan herhangi bir sayı olacaktır

Herbir sembolü işlemek için, geçerli aralık küçültülür ve bu da kodun daha fazla bitle ifade edilmesine sebep olur

Sonuç : Tek bir sayıdır ve bu sayı tek bir sembolü değil, bütün input stream'i temsil eder.

Ort.kod uzunluğu = output bit uzunluğu / input stream'deki sembol sayısı

- Sıkıştırılmış veri output stream'e yazılmadan önce, sembol ve onların sıklık değerleri çıkışa yazılır
- Bu bölüm kod çözücü tarafından okunacak ilk şeydir

- Kodlama işlemi LOW ve HIGH isimli iki değişkenin tanımlanması ile başlar
- Değişkenlere 0 ve 1 değerleri atanır
- Aralık [LOW, HIGH) olarak gösterilir
- Semboller okunup işlendikten sonra LOW ve HIGH değerleri birbirlerine yakınlaşır ve aralık daraltılır

Örnek

Char	Freq	Prob.	Range	CumFreq
		Total CumFreq=		10
S	5	$5/10 = 0.5$	$[0.5, 1.0)$	5
W	1	$1/10 = 0.1$	$[0.4, 0.5)$	4
I	2	$2/10 = 0.2$	$[0.2, 0.4)$	2
M	1	$1/10 = 0.1$	$[0.1, 0.2)$	1
␣	1	$1/10 = 0.1$	$[0.0, 0.1)$	0

Kümülatif frekans (CumFreq) kolonu kod çözme algoritması tarafından kullanılmaktadır.

NewHigh = OldLow + Range * HighRange(x)
NewLow = OldLow + Range * LowRange(x)

Range = OldHigh – OldLow

LowRange(x) ve HighRange(x) x sembolünün alt ve üst limit değerleridir

Char.		The calculation of low and high
S	L	$0.0 + (1.0 - 0.0) \times 0.5 = 0.5$
	H	$0.0 + (1.0 - 0.0) \times 1.0 = 1.0$
W	L	$0.5 + (1.0 - 0.5) \times 0.4 = 0.70$
	H	$0.5 + (1.0 - 0.5) \times 0.5 = 0.75$
I	L	$0.7 + (0.75 - 0.70) \times 0.2 = 0.71$
	H	$0.7 + (0.75 - 0.70) \times 0.4 = 0.72$
S	L	$0.71 + (0.72 - 0.71) \times 0.5 = 0.715$
	H	$0.71 + (0.72 - 0.71) \times 1.0 = 0.72$
S	L	$0.715 + (0.72 - 0.715) \times 0.5 = 0.7175$
	H	$0.715 + (0.72 - 0.715) \times 1.0 = 0.72$
␣	L	$0.7175 + (0.72 - 0.7175) \times 0.0 = 0.7175$
	H	$0.7175 + (0.72 - 0.7175) \times 0.1 = 0.71775$
M	L	$0.7175 + (0.71775 - 0.7175) \times 0.1 = 0.717525$
	H	$0.7175 + (0.71775 - 0.7175) \times 0.2 = 0.717550$
I	L	$0.717525 + (0.71755 - 0.717525) \times 0.2 = 0.717530$
	H	$0.717525 + (0.71755 - 0.717525) \times 0.4 = 0.717535$
S	L	$0.717530 + (0.717535 - 0.717530) \times 0.5 = 0.7175325$
	H	$0.717530 + (0.717535 - 0.717530) \times 1.0 = 0.717535$
S	L	$0.7175325 + (0.717535 - 0.7175325) \times 0.5 = 0.71753375$
	H	$0.7175325 + (0.717535 - 0.7175325) \times 1.0 = 0.717535$

Char	Freq	Prob.	Range	CumFreq
		Total CumFreq=		10
S	5	$5/10 = 0.5$	[0.5, 1.0)	5
W	1	$1/10 = 0.1$	[0.4, 0.5)	4
I	2	$2/10 = 0.2$	[0.2, 0.4)	2
M	1	$1/10 = 0.1$	[0.1, 0.2)	1
␣	1	$1/10 = 0.1$	[0.0, 0.1)	0

- Kod çözücü karşı yönde çalışır.
- Kod çözücü, sembol ve onların aralık değerlerini okuyarak, başlangıçta oluşturulmuş olan tabloyu tekrardan oluşturur (char, freq, prob, range, cumfreq)
- İlk dijit olarak 7'yi okur ve bilirki bu değer 0,7'dir

Char.	Code−low	Range
S	$0.71753375 - 0.5 = 0.21753375$	$/0.5 = 0.4350675$
W	$0.4350675 - 0.4 = 0.0350675$	$/0.1 = 0.350675$
I	$0.350675 - 0.2 = 0.150675$	$/0.2 = 0.753375$
S	$0.753375 - 0.5 = 0.253375$	$/0.5 = 0.50675$
S	$0.50675 - 0.5 = 0.00675$	$/0.5 = 0.0135$
␣	$0.0135 - 0 = 0.0135$	$/0.1 = 0.135$
M	$0.135 - 0.1 = 0.035$	$/0.1 = 0.35$
I	$0.35 - 0.2 = 0.15$	$/0.2 = 0.75$
S	$0.75 - 0.5 = 0.25$	$/0.5 = 0.5$
S	$0.5 - 0.5 = 0$	$/0.5 = 0$

Char	Freq	Prob.	Range	CumFreq
		Total CumFreq=		10
S	5	$5/10 = 0.5$	[0.5, 1.0)	5
W	1	$1/10 = 0.1$	[0.4, 0.5)	4
I	2	$2/10 = 0.2$	[0.2, 0.4)	2
M	1	$1/10 = 0.1$	[0.1, 0.2)	1
␣	1	$1/10 = 0.1$	[0.0, 0.1)	0

Char	Prob.	Range
a_1	0.001838	[0.998162, 1.0)
a_2	0.975	[0.023162, 0.998162)
a_3	0.023162	[0.0, 0.023162)

Encoding the String $a_2a_2a_1a_3a_3$.

a_2	$0.0 + (1.0 - 0.0) \times 0.023162 = 0.023162$
	$0.0 + (1.0 - 0.0) \times 0.998162 = 0.998162$
a_2	$0.023162 + .975 \times 0.023162 = 0.04574495$
	$0.023162 + .975 \times 0.998162 = 0.99636995$
a_1	$0.04574495 + 0.950625 \times 0.998162 = 0.9946270125$
	$0.04574495 + 0.950625 \times 1.0 = 0.99636995$
a_3	$0.9946270125 + 0.00174724875 \times 0.0 = 0.9946270125$
	$0.9946270125 + 0.00174724875 \times 0.023162 = 0.994663171025547$
a_3	$0.9946270125 + 0.00004046977554749998 \times 0.0 = 0.9946270125$
	$0.9946270125 + 0.00004046977554749998 \times 0.023162 = 0.994623638610941$

Char.	Code−low		Range	
a_2	$0.9946270125 - 0.023162$	$= 0.9714617012 5$	$/0.975$	$= 0.99636995$
a_2	$0.99636995 - 0.023162$	$= 0.97320795$	$/0.975$	$= 0.998162$
a_1	$0.998162 - 0.998162$	$= 0.0$	$/0.00138$	$= 0.0$
a_3	$0.0 - 0.0$	$= 0.0$	$/0.023162$	$= 0.0$
a_3	$0.0 - 0.0$	$= 0.0$	$/0.023162$	$= 0.0$

Bu problem, input stream'de en son yer alan sembolün alt aralık sınırı 0 dan başlıyorsa meydana gelir

Char	Prob.	Range
eof	0.000001	[0.999999, 1.0)
a_1	0.001837	[0.998162, 0.999999)
a_2	0.975	[0.023162, 0.998162)
a_3	0.023162	[0.0, 0.023162)

$a_3 a_3 a_3 a_3$eof.

a_3	$0.0 + (1.0 - 0.0) \times 0.0 = 0.0$
	$0.0 + (1.0 - 0.0) \times 0.023162 = 0.023162$
a_3	$0.0 + .023162 \times 0.0 = 0.0$
	$0.0 + .023162 \times 0.023162 = 0.000536478244$
a_3	$0.0 + 0.000536478244 \times 0.0 = 0.0$
	$0.0 + 0.000536478244 \times 0.023162 = 0.000012425909087528$
a_3	$0.0 + 0.000012425909087528 \times 0.0 = 0.0$
	$0.0 + 0.000012425909087528 \times 0.023162 = 0.000000287808906285323 5$
eof	$0.0 + 0.000000287808906285323 5 \times 0.999999 = 0.000000287808618476417 2$
	$0.0 + 0.000000287808906285323 5 \times 1.0 = 0.000000287808906285323 5$

Ek bir EOF sembolü kullanmak gerekir veya kodlayıcı output sream'e önce input stream'in boyutunu yazar ve kontrolü böyle sağlar

Char.	Code−low		Range
a_3	0.000000287808618476417 2-0	=0.000000287808618476417 2	/0.023162=0.000012425896661618912 47
a_3	0.000012425896661618912 47-0	=0.000012425896661618912 47	/0.023162=0.000536477707521756
a_3	0.000536477707521756-0	=0.000536477707521756	/0.023162=0.023161976838
a_3	0.023161976838-0.0	=0.023161976838	/0.023162=0.999999
eof	0.999999-0.999999	=0.0	/0.000001=0.0

- Yapılan bu işlemler pratik değildir
- Tek bir sayı olan kod uzundur
- 1MB bir dosya, tek bir sayı ile (500KB) kodlanır

Aritmetik Kodlamada pratik uygulama için integer sayı kullanılır

- Vereceğimiz örnekte HIGH ve LOW isimli değişkenler 4 dijit uzunluğunda olsun
- Bir önceki uygulamada HIGH ve LOW değişkenler görülmektedir ve değişkenlerin en solundaki değerler hiç değişmemektedir
- Değişkenlerin, dijitlerini öteleyerek output stream'e tek bir dijit yazılabilir. Böylece her iki değişkende kodun tamamını tutmaz.
- Dijitleri ötelerken Low değişkenine 0, High değişkenine de sağdan 9 girilir

«SWISS MISS» stringinin kodlama işlemini gösterelim

1.kolon okunacak input stream
2.kolon LOW ve HIGH yeni değerleri
3.kolon bu değerlerin ölçeklendirilmiş hali
(LOW ve HIGH değerleri 10bin ile çarpılır, HIGH değerinden 1 çıkarılır
4.kolon, output streame gönderilecek dijit
5.kolon sola ötelemeden sonraki yeni
LOW ve HIGH değerleri

Output stream'e «717533750» değeri gönderilir

1	2	3	4	5
S	$L = 0+(1-0)\times 0.5 = 0.5$	5000		5000
	$H = 0+(1-0)\times 1.0 = 1.0$	9999		9999
W	$L = 0.5+(1-.5)\times 0.4 = 0.7$	7000	7	0000
	$H = 0.5+(1-.5)\times 0.5 = 0.75$	7499	7	4999
I	$L = 0+(0.5-0)\times 0.2 = 0.1$	1000	1	0000
	$H = 0+(0.5-0)\times 0.4 = 0.2$	1999	1	9999
S	$L = 0+(1-0)\times 0.5 = 0.5$	5000		5000
	$H = 0+(1-0)\times 1.0 = 1.0$	9999		9999
S	$L = 0.5+(1-0.5)\times 0.5 = 0.75$	7500		7500
	$H = 0.5+(1-0.5)\times 1.0 = 1.0$	9999		9999
␣	$L = 0.75+(1-0.75)\times 0.0 = 0.75$	7500	7	5000
	$H = 0.75+(1-0.75)\times 0.1 = 0.775$	7749	7	7499
M	$L = 0.5+(0.75-0.5)\times 0.1 = 0.525$	5250	5	2500
	$H = 0.5+(0.75-0.5)\times 0.2 = 0.55$	5499	5	4999
I	$L = 0.25+(0.5-0.25)\times 0.2 = 0.3$	3000	3	0000
	$H = 0.25+(0.5-0.25)\times 0.4 = 0.35$	3499	3	4999
S	$L = 0+(0.5-0)\times 0.5 = .25$	2500		2500
	$H = 0+(0.5-0)\times 1.0 = 0.5$	4999		4999
S	$L = 0.25+(0.5-0.25)\times 0.5 = 0.375$	3750		3750
	$H = 0.25+(0.5-0.25)\times 1.0 = 0.5$	4999		4999

Encoding SWISS␣MISS by Shifting.

Kod çözme işlemi LOW=0000 HIGH=9999 ve Code = 7175 atanarak işleme başlanır.
Index deki 10 değeri Toplam CumFreq değeridir.

```
Low:=Low+(High-Low+1)LowCumFreq[X]/10;
High:=Low+(High-Low+1)HighCumFreq[X]/10-1;

index:=((Code-Low+1)x10-1)/(High-Low+1)
```

0. Initialize Low=0000, High=9999, and Code=7175.

1. index= $[(7175-0+1)\times 10-1]/(9999-0+1)=7.1759\rightarrow 7$. Symbol S is selected.
Low $=0+(9999-0+1)\times 5/10=5000$. High $=0+(9999-0+1)\times 10/10-1=9999$.

2. index= $[(7175-5000+1)\times 10-1]/(9999-5000+1)=4.3518\rightarrow 4$. Symbol W is selected.
Low $=5000+(9999-5000+1)\times 4/10=7000$. High $=5000+(9999-5000+1)\times 5/10-1=7499$.
After the 7 is shifted out, we have Low=0000, High=4999, and Code=1753.

3. index= $[(1753-0+1)\times 10-1]/(4999-0+1)=3.5078\rightarrow 3$. Symbol I is selected.
Low $=0+(4999-0+1)\times 2/10=1000$. High $=0+(4999-0+1)\times 4/10-1=1999$.
After the 1 is shifted out, we have Low=0000, High=9999, and Code=7533.

4. index= $[(7533-0+1)\times 10-1]/(9999-0+1)=7.5339\rightarrow 7$. Symbol S is selected.
Low $=0+(9999-0+1)\times 5/10=5000$. High $=0+(9999-0+1)\times 10/10-1=9999$.

5. index= $[(7533-5000+1)\times 10-1]/(9999-5000+1)=5.0678\rightarrow 5$. Symbol S is selected.
Low $=5000+(9999-5000+1)\times 5/10=7500$. High $=5000+(9999-5000+1)\times 10/10-1=9999$.

```
Low:=Low+(High-Low+1)LowCumFreq[X]/10;
High:=Low+(High-Low+1)HighCumFreq[X]/10-1;
```

```
index:=((Code-Low+1)x10-1)/(High-Low+1)
```

6. index= $[(7533-7500+1)\times 10-1]/(9999-7500+1)=0.1356\rightarrow 0$. Symbol ␣ is selected.
Low $=7500+(9999-7500+1)\times 0/10=7500$. High $=7500+(9999-7500+1)\times 1/10-1=7749$.
After the 7 is shifted out, we have Low=5000, High=7499, and Code=5337.

7. index= $[(5337-5000+1)\times 10-1]/(7499-5000+1)=1.3516\rightarrow 1$. Symbol M is selected.
Low $=5000+(7499-5000+1)\times 1/10=5250$. High $=5000+(7499-5000+1)\times 2/10-1=5499$.
After the 5 is shifted out we have Low=2500, High=4999, and Code=3375.

8. index= $[(3375-2500+1)\times 10-1]/(4999-2500+1)=3.5036\rightarrow 3$. Symbol I is selected.
Low $=2500+(4999-2500+1)\times 2/10=3000$. High $=2500+(4999-2500+1)\times 4/10-1=3499$.
After the 3 is shifted out we have Low=0000, High=4999, and Code=3750.

9. index= $[(3750-0+1)\times 10-1]/(4999-0+1)=7.5018\rightarrow 7$. Symbol S is selected.
Low $=0+(4999-0+1)\times 5/10=2500$. High $=0+(4999-0+1)\times 10/10-1=4999$.

10. index= $[(3750-2500+1)\times 10-1]/(4999-2500+1)=5.0036\rightarrow 5$. Symbol S is selected.
Low $=2500+(4999-2500+1)\times 5/10=3750$. High $=2500+(4999-2500+1)\times 10/10-1=4999$.

```
Low:=Low+(High-Low+1)LowCumFreq[X]/10;
High:=Low+(High-Low+1)HighCumFreq[X]/10-1;
```

index:=((Code-Low+1)x10-1)/(High-Low+1)

Pratikte bütün işlemler binary sayılar üzerinde gerçekleşir.

«SWISS MISS» stringinin LOW = 0.71753375 HIGH = 0.717535

Yaklaşık binary değeri
L = 0.10110111101100000100101010111
H = 0.10110111101100000100111111011

10 sembol 29 bitle yaklaşık 4 byte ile sıkıştırılıyor.

Prediction with Partial String Matching - PPM

- J.Cleary, I. Witten tarafından ortaya atılmış, A.Moffat tarafından da daha iyi hale getirilmiştir
- PPM-Prediction with Partial String Matching, «order-N» Markov modelini kullanan içerik tabanlı bir sıkıştırma yöntemidir
- Model, sembole bir olasılık değeri atamadan evvel sembolün içeriğini inceler
- Kodlayıcı ve kod çözücü, text'in ilerisine erişemediğinden, her ikisi de geçmişteki text'in içeriği ile ilgilenir
- Pratikte bir sembolün içeriği kendisinden önce gelen N adet semboldür

İstatistiksel Model

Text'in geçmişinde bulunan her sembolün kullanım sıklığı belirlenir ve bu değere bağlı olarak sembolün olasılığı bulunur

- 1217 adet sembolün, 34 tanesi **q** olsun
- Sembolün olasılığı **34/1217**
- Gelen yeni bir **q** değeri ile olasılık **35/1217** olarak değişir

İçerik Tabanlı İstatistiksel Model

- Bir **S** sembolüne, sembolün sıklığına bağlı olmaksızın, context içerisindeki sıklığına bakılarak bir olasılık değeri atanır

h→%5
t→%30

th→ olasılığı çok daha fazladır

Statik İçerik Tabanlı Yöntemler

- Her zaman aynı olasılık değerleri kullanılır. Alfabedeki mümkün olan bütün di-gram ve tri-gram olasılık değerlerinden statik bir tablo oluşturulur

- Bir **S** sembolü okunduğunda, kendisinden önce okunmuş **C** context'ine bağlı olarak tablodan olasılık değerleri çekilerek kullanılır

Adaptive İçerik Tabanlı Yöntemler

- Text'deki mümkün olan bütün di-gram/tri-gram'ların olasılık değerleri ile tablolar oluşturulur
- Yeni okunan bir **S** sembolünün olasılığı, kendisinden önce gelen birkaç sembole bağlı olarak bu tablolar kullanılarak belirlenir ve tablolar sürekli olarak güncellenir
- Yavaş ve karışık bir yöntem olmasına karşılık daha iyi sıkıştırma oranı elde edilir

Lipogram Nedir ?

Bir text'e kullanılan karakterlerin olasılık değerleri birbirlerinden çok ayrık ise bu durum «lipogram» olarak adlandırılır

- En iyi bilinen lipogram *Ernest Vimcent Wright*

  Gadsby → İçerisinde E harfi yoktur

- Walter Abish «Alphabetical Africa»

  52 bölümden oluşur

  1.Bölümde bütün kelimeler a ile başlar
  2.bölümde bütün kelimeler a ve b ile başlar
  ……
  26.bölümde bütün kelimeler a,…,z ile başlar
  Geriye kalan 26 bölümde aynı şekilde azalarak devam eder

Order-N Adaptive Context_Based Model

- Input stream'den gelecek olan **S** sembolünü okur ve mevcut order-N modeline göre kendisinden önce gelmiş olan N sembolü bulur ve S'nin olasılığını tahmin eder
- Teoride N değeri çok büyük seçilir (N=20 bin) ancak pratikte bunun dezavantajı vardır

Eğer bir sembolü kendisinden önce gelen 20 bin sembole bağlı olarak kodlayacak isek ilk 20 bin sembol nasıl kodlanmalıdır ?

- Output stream'e ASCII kodları yazılır
- N'nin büyük değerlerinde, çok fazla sayıda context oluşur

- Eğer sembollerimiz 7 bit ASCII kodu ise, alfabede 2^7=128 sembol olup, 128^2=1.684 tane order-2
- 128^3=2.097.152 tane order-3 olur

- Çok uzun context'lerde eski bilgilerde elde tutulur. Çok geniş veri dosyalarında, farklı bölümlerde sembollerin dağılımları da farklı olabilir (tarih kitapları gibi)

Sonuç

Pratikte N-Context modelinde N değeri 2 ile 10 arasında sınırlı olmalıdır

Order 4	Order 3	Order 2	Order 1	Order 0
xyzz→x 2	xyz→z 2	xy→z 2	x→y 3	x 4
yzzx→y 1	yzz→x 2	→x 1	y→z 2	y 3
zzxy→x 1	zzx→y 1	yz→z 2	→x 1	z 4
zxyx→y 1	zxy→x 1	zz→x 2	z→z 2	
xyxy→z 1	xyx→y 1	zx→y 1	→x 2	
yxyz→z 1	yxy→z 1	yx→y 1		

(a)

Order 4	Order 3	Order 2	Order 1	Order 0
xyzz→x 2	xyz→z 2	xy→z 2	x→y 3	x 4
yzzx→y 1	yzz→x 2	xy→x 1	→z 1	y 3
→z 1	zzx→y 1	yz→z 2	y→z 2	z 5
zzxy→x 1	→z 1	zz→x 2	→x 1	
zxyx→y 1	zxy→x 1	zx→y 1	z→z 2	
xyxy→z 1	xyx→y 1	→z 1	→x 2	
yxyz→z 1	yxy→z 1	yx→y 1		

(b)

Table 2.72: (a) Contexts and Counts for "xyzzxyxyzzx". (b) Updated After Another z Is Input.

PPM'in Çalışma Prensibi

- PPM, sıfır olasılıklı bir context geldiğinde, onu daha kısa uzunluktaki bir context ile yer değiştirir (switching)
- PPM, order-N context ile başlar. Gelen bir S sembolünü order-N modeline göre önceden gelmiş olan C context'i içerisinde arar
- Eğer bulamaz ise (**S** sembolünün izlediği bir **C** context'i mevcut değilse) Order-N-1'e switch eder
- PPM kodlayıcısı, Order-0 context'ine switch ettiğinde «S» sembolünün geçmişte kaç kez göründüğünde bakar, daha önceden hiç görülmemiş ise «S» için sabit olasılık bir değer atanır (1/alfabenin boyu)
- Bulunan **P** olasılıklı her **S** sembolü kodlanmak için Arithmetic Coding algoritmasına gönderilir

PPM'in Kod Çözücü

- PPM kod çözücü, kodu çözerken hangi aşamada switching işleminin gerçekleştiğini bilmesi gerekir
- Bunun için **escape** sembolü kullanır
- Kodlayıcı, switch işlemini gerçekleştirmeden önce **escape** sembolünün olasılığını Arithmetic Coding gönderir

Order 2			Order 1			Order 0		
Context	f	p	Context	f	p	Symbol	f	p
as→s	2	2/3	a→ s	2	2/5	a	4	4/19
esc	1	1/3	a→ n	1	1/5	s	6	6/19
			esc→	2	2/5	n	1	1/19
ss→a	2	2/5				i	2	2/19
ss→i	1	1/5	s→ s	3	3/9	m	1	1/19
esc	2	2/5	s→ a	2	2/9	esc	5	5/19
			s→ i	1	1/9			
sa→n	1	1/2	esc	3	3/9			
esc	1	1/2						
			n→ i	1	1/2			
an→i	1	1/2	esc	1	1/2			
esc	1	1/2						
			i→ s	1	1/4			
ni→s	1	1/2	i→ m	1	1/4			
esc	1	1/2	esc	2	2/4			
is→s	1	1/2	m→ a	1	1/2			
esc	1	1/2	esc	1	1/2			
si→m	1	1/2						
esc	1	1/2						
im→a	1	1/2						
esc	1	1/2						
ma→s	1	1/2						
esc	1	1/2						

Table 2.73: Contexts, Counts (f), and Probabilities (p) for "as-sanissimassa".

Bazı senaryolar yaratalım. Son context değerimiz order-2 ile **«sa»** olsun

1. Gelecek sembol **«n»** olsun
   «n» nin takip ettiği **«sa»** context'i bulunur
   Olasılığı $\frac{1}{2}$ dir
   «n» bu olasılık değeri ile Aritmetik Kodlamaya gönderilir $-\log_2(1/2) = 1$ bit

2. Gelecek sembolün **«s»** olduğunu kabul edelim
   «s» nin izlediği **«sa»** context'i aranır (order-2)
   Bulamayınca $\frac{1}{2}$ olasılıkla **«escape»** sembolünü gönderir
   Order-1'e switch eder
   «s» nin izlediği **«a»** aranır ve 2/5 olasılıkla bulunur
   $\log_2(1/2) + \log_2(2/5) = 2.32$ bit

3. Gelecek sembolün **«m»** olduğunu kabul edelim

«m» nin izlediği **«sa»** context'i aranır (order-2)

Bulamayınca 1 bitlik **«escape»** sembolünü gönderir

Order-1'e switch eder

«m» nin izlediği **«a»** contex'i order-1 de aranır

Bulamayınca $-\log_2(2/5) = 1.32$ bitlik **«escape»** sembolünü gönderir

Order-0'a switch eder

«m» olasılığı 1/19 bulunarak - $\log_2(1/19) = 4.25$ bit ile kodlanır

Toplam 1 + 1.32 + 4.25 = 6.57 bit kodlanır

4. Gelecek sembolün **«d»** olduğunu kabul edelim

order-2 $-\log_2(1/2) = 1$ bit

order-1 $-\log_2(2/5) = 1.32$ bit

order-0 $-\log_2(5/19) = 1.93$ bit

order- -1 «d» 1/28 olasılıkla $-\log_2(1/28) = 4.8$ bit

Toplam 9.05 bit gerekir

PPM'in üç farklı versiyonu vardır. Bu modellerin sadece «escape» sembollerinin kullanımında farklılıklar vardır. Sıkıştırma oranlarında da çok küçük farklar oluşmaktadır.

Bir önceki gördüğümüz yöntem PPMC

PPMA'da, toplam frekans «n» ise «escape» sembolüne 1/(n+1) olasılığı, diğer sembollere de x/n olasılığı atanır

PPMB'de, ***C*** context'inin içeriğinde «s» iki kez gözüktükten sonra, ***C*** context'ini izleyen **S** sembolüne bir olasılık atanır.

Eğer «abc» context'i üç kez ve bunu izleyen «x» iki kez «y» de bir kez gözükmüş ise:

X için (2-1)/3
Y için de (1-1)/3 olasılığı atanır

- PPM'in pratikteki uygulamasında ana problem input stream'den okunan her sembolün 0'dan N'ye kadar bütün context'lerinin veri yapısını oluşturmak ve bunlara hızlı bir şekilde erişmektir
- «Trie» adı verilen özel bir veri yapısı kullanılır. Ağacın her bir dalı birbirine yakın context'lerden oluşur.

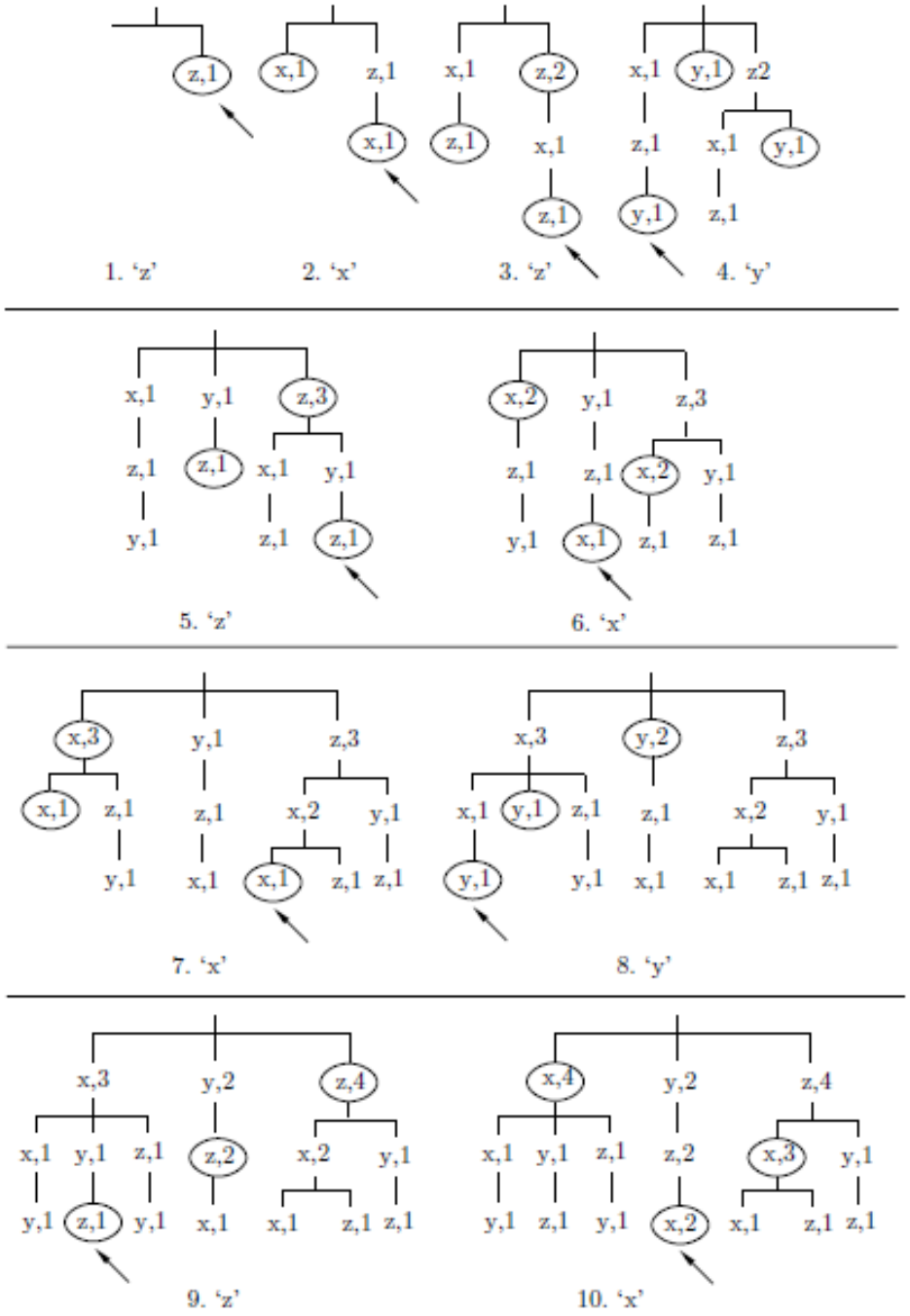


Figure 2.74: Ten Tries of "zxzyzxxyzx".

- Algoritmayı basitleştirmek için, herbir düğüme bir pointer eklenir
- Bu pointer geriye doğru bir sonraki kısa context'i gösterir
- Ağaçta geriyi işaret eden bu pointer «wine pointer» olarak adlandırılır

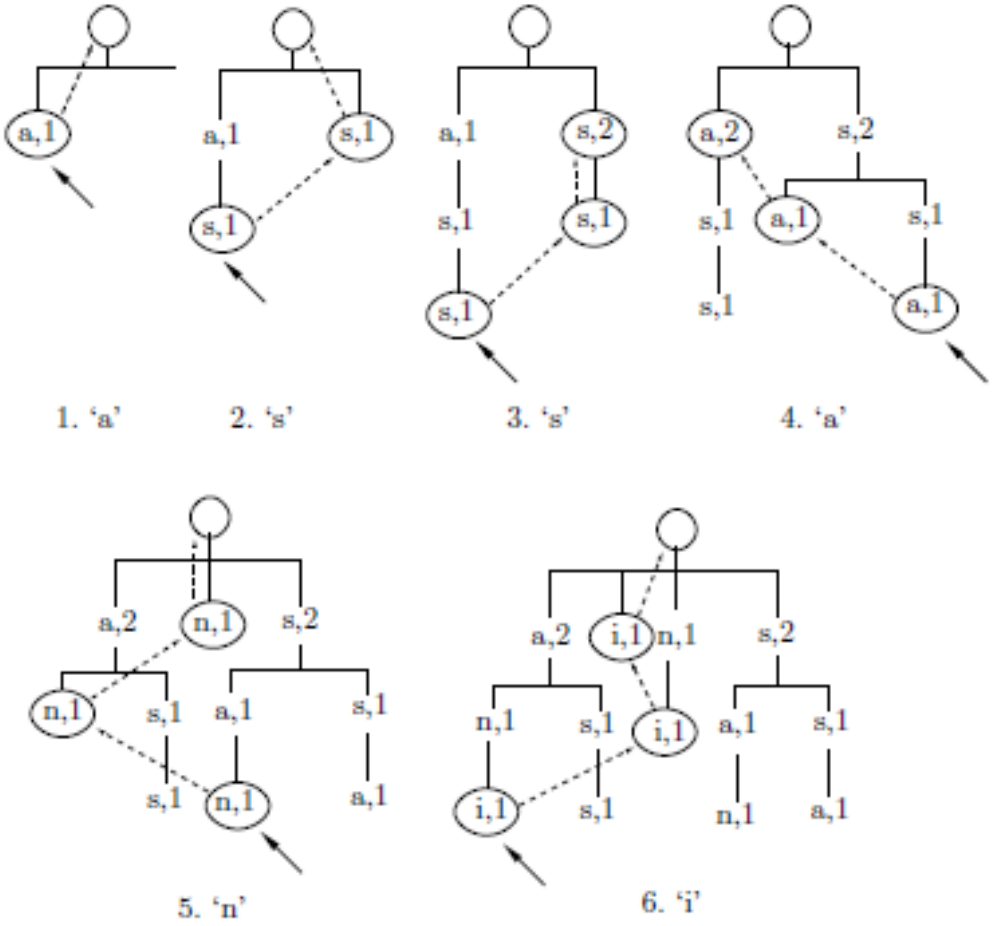


Figure 2.75: Part I. First Six Tries of "assanissimassa".

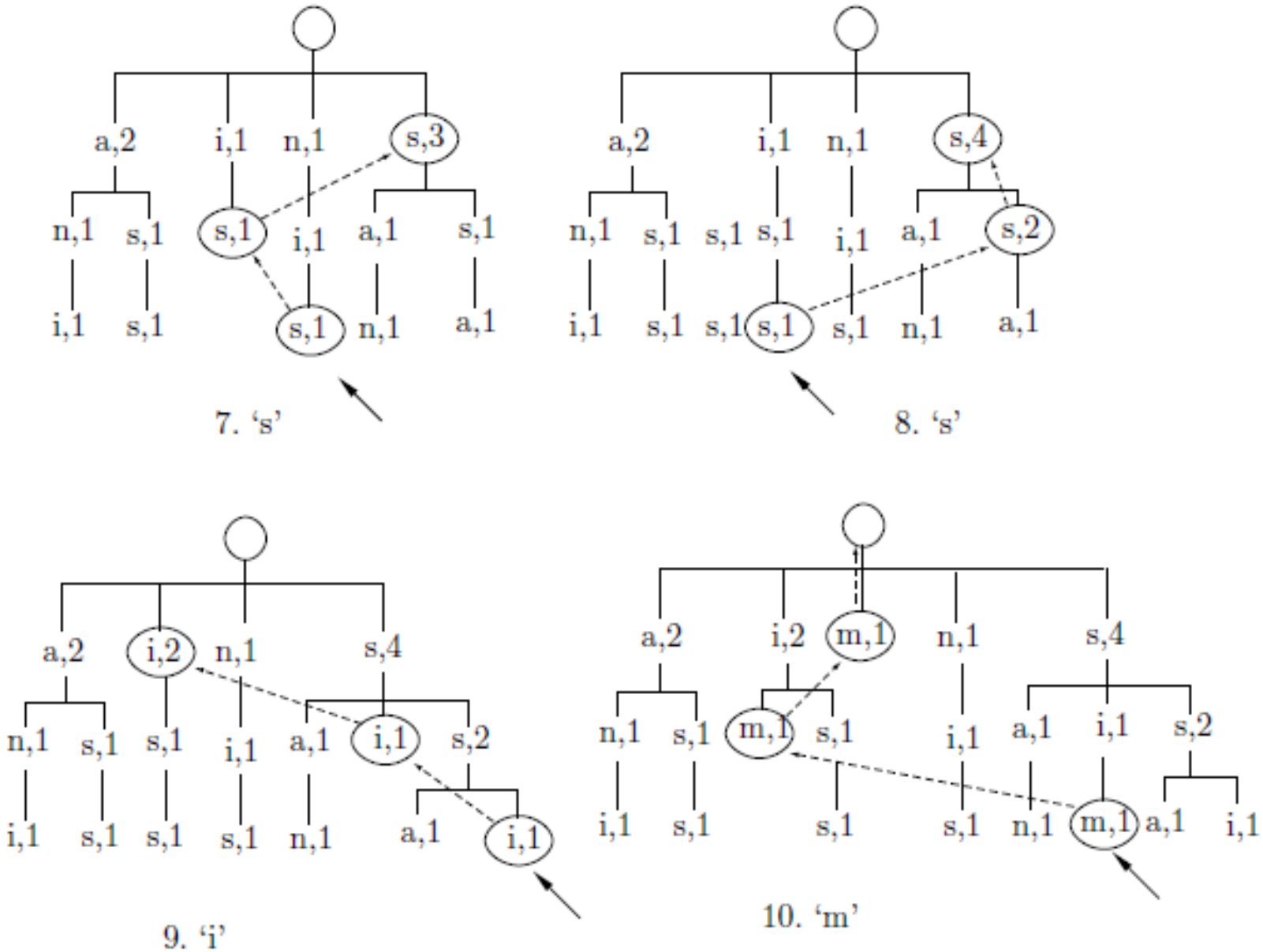
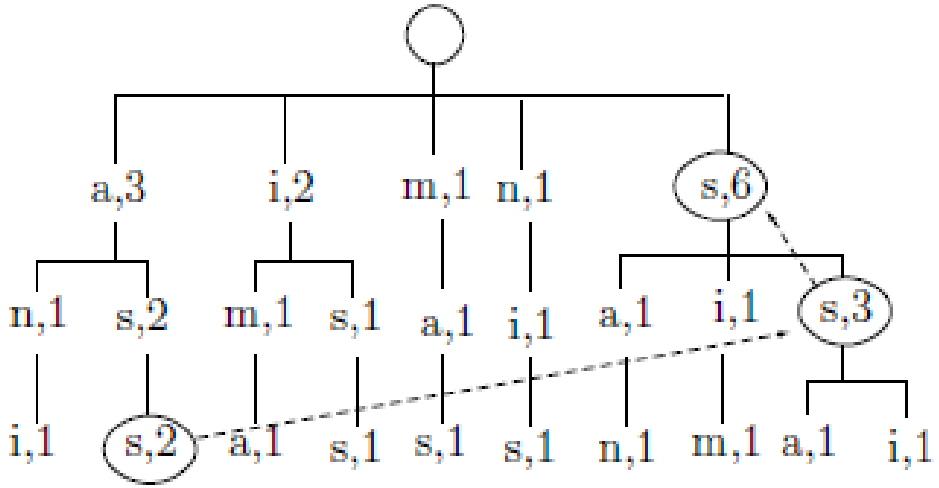
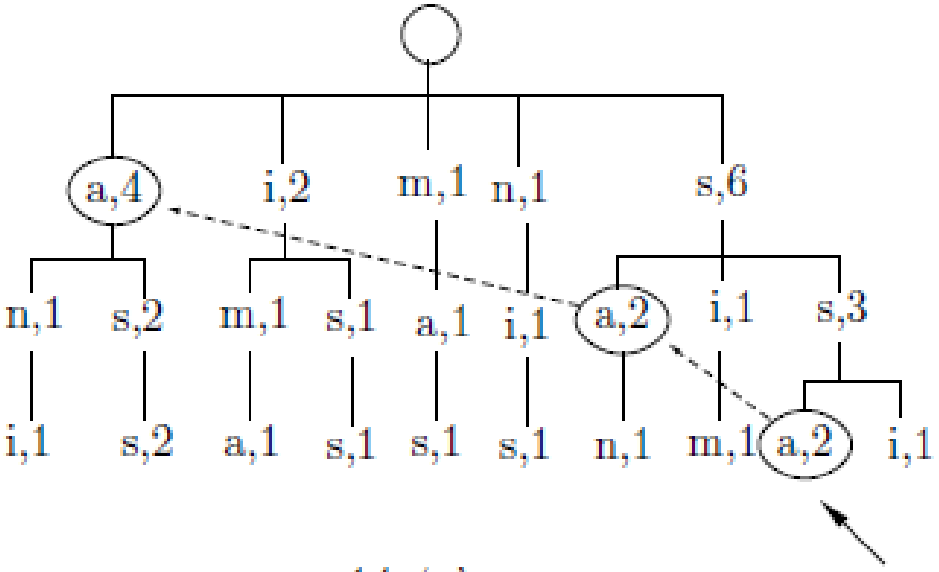


Figure 2.75: (Continued) Next Four Tries of "assanissimassa".



13. 's'



14. 'a'

Figure 2.75: (Continued) Final Two Tries of "assanissimassa".